BALLUFF

stdlib und Abstraktion

Bezug zur newlib auf dem STM32

DHBW

Ein paar Problemfälle

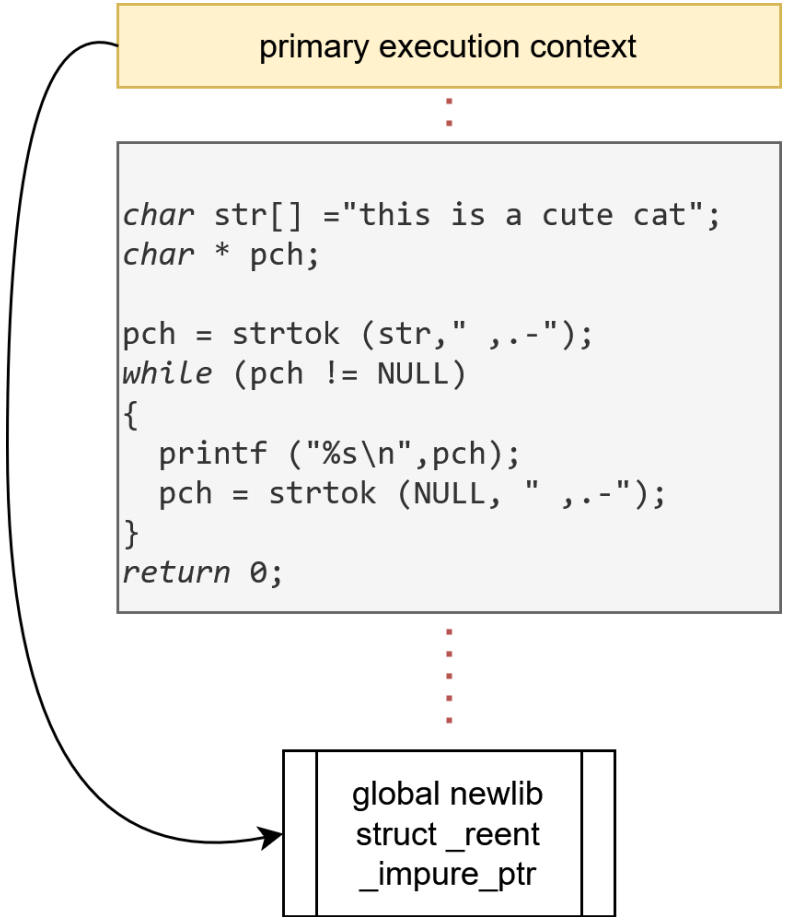
- (*malloc*) Speicherallokation schlägt sporadisch fehl und die Anwendung hängt sich auf
- (*setenv*) Umgebungsvariablen gehen nach dem Schreiben neuer Variablen verloren
- (*stdlib*) Nach einer Ausführungsdauer von 5 Stunden stürzt meine Anwendung ab
- (*strtok*) Beim Zerlegen von Zeichenketten stürzt meine Anwendung sporadisch ab oder verliert einzelne Tokens
- (*printf*) Beim Ausgeben von Zeichenketten funktioniert meine Konsolenausgabe sporadisch nicht mehr

stdlib-Nutzung

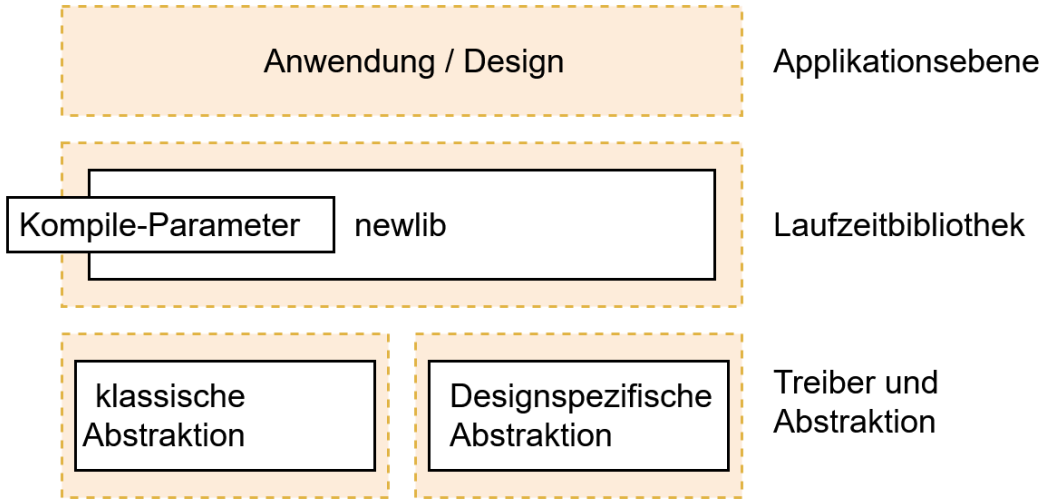
- In der Regel kommt in eingebetteten Systemen die newlib von Red Hat zum Einsatz, welche für Single-Threaded Designs bereits korrekt portiert durch Microcontroller-Hersteller bereitgestellt wird
- Bei Multi-Threaded Designs haben die mitgelieferten Portierungen erhebliche Fehler und sind teilweise nicht für den Produktiveinsatz geeignet, da grob fahrlässige Fehler enthalten sind (Codegeneratoren)
- Multi-Threading erfordert separate Portierungs- und Abstraktionsschritte der newlib

stdlib-Nutzung

- Alle Funktionen der stdlib nutzen eine globale interne Zustandsstruktur
  - Sofern ein Zustand gehalten werden muss
  - In dieser Strutkur sind auch stdio Streams definiert (stdin, stdout, stderr)



Klassische Abstraktion

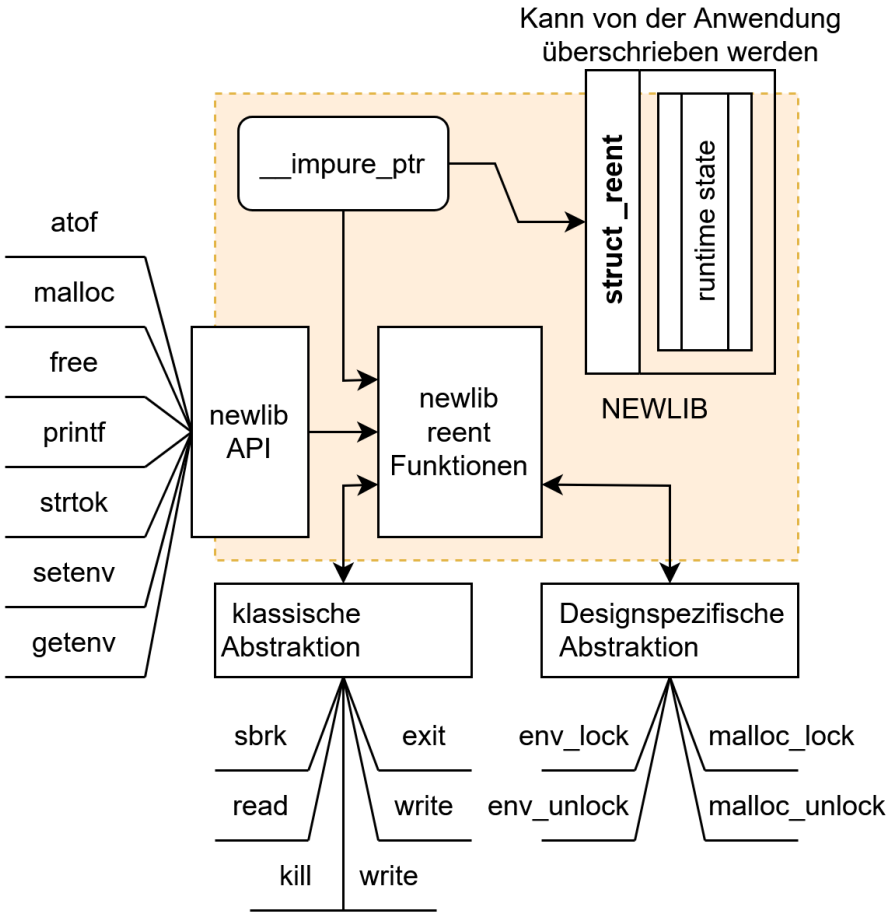


- Auf mehreren Ebenen findet eine Abstraktion der newlib statt.
  - Die Compilertoolchain bringt vorkompilierte stdlib Archive mit, welche bereits gewisse Randparameter fest einkompiliert haben (nanolib, float-support, speed/size-Optimierung, Featurereichtum)
  - Die klassischen Abstraktionsfunktionen im Design oder durch mitgelieferten Startcode der Toolchain (sbrk, crtbegin.S, crtend.S, crt0.S, …)
  - Zusätzliche Abstraktionsfunktionen und Lock-Mechanismen

Die wichtigsten klassischen Abstraktionen

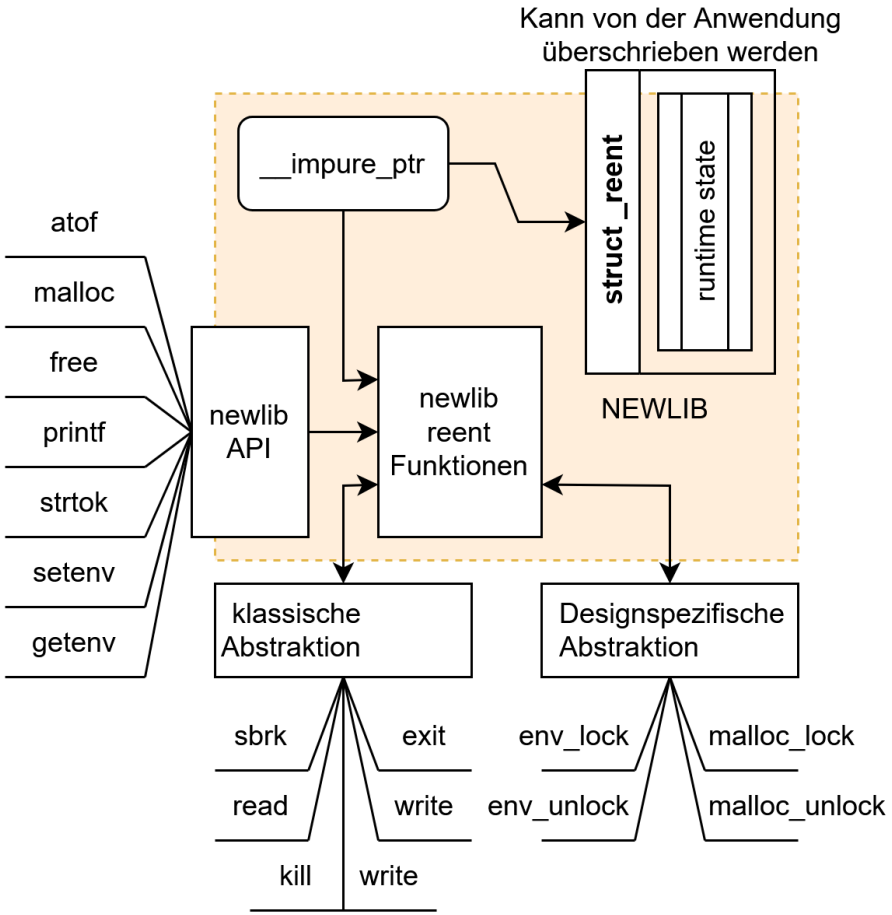
- Bei einem STM32 sind teilweise die folgenden Funktionen mit Cube generiert:

- _sbrk
- _exit
- _isatty
- _read
- _write



Die wichtigsten spezifischen Abstraktionen

- env_lock
- env_unlock
- malloc_lock
- malloc_unlock



Welche Funktionen sind kritisch

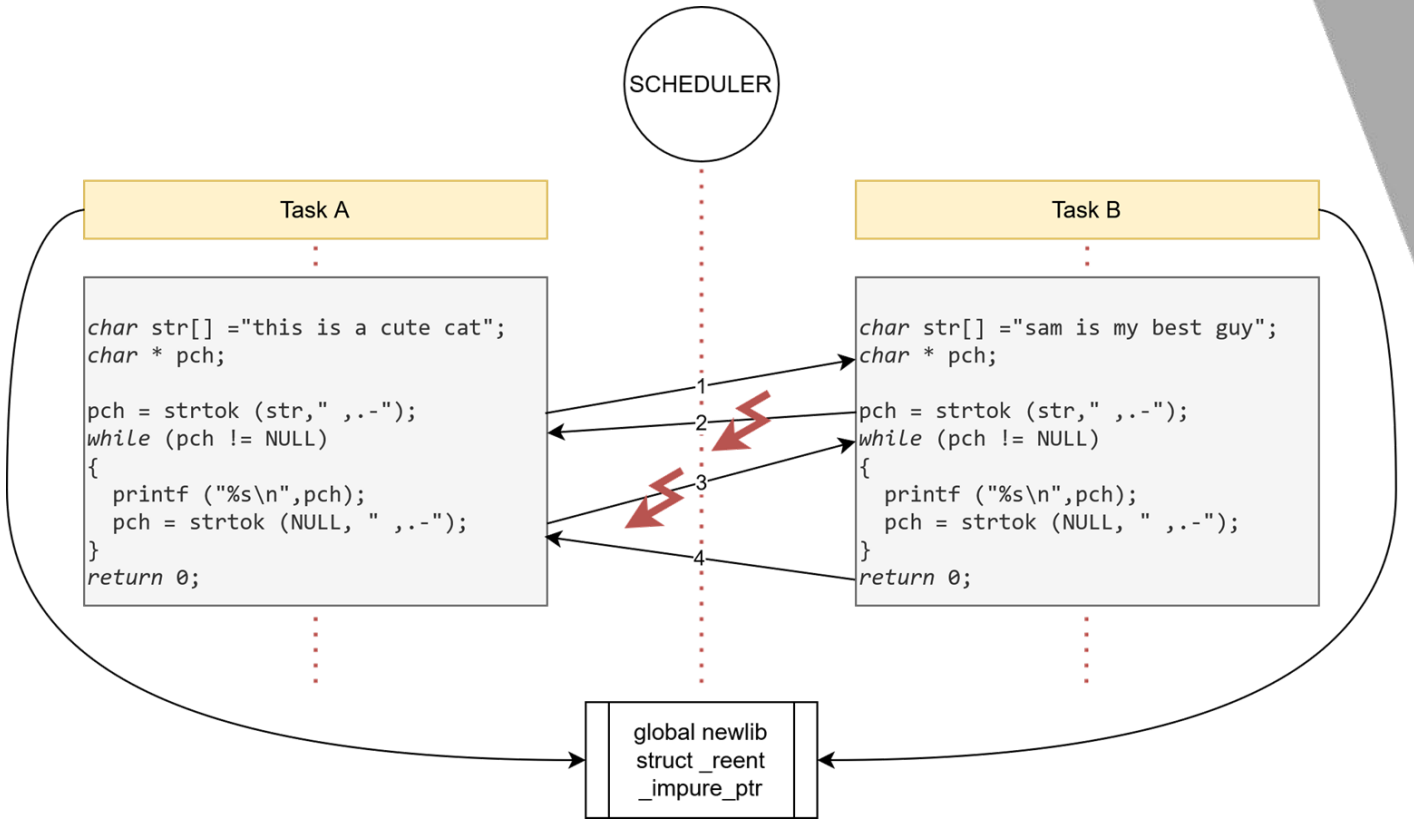
- Zustandsspeichernde Funktionen wie bspw. *strtok*
- Funktionen mit Nutzung von dynamischem Speicher wie bspw. *malloc, printf, atof*
- Solange nur ein Ausführungskontext vorliegt, ist keine willkürliche Unterbrechbarkeit gegeben (preemption) [keine stdlib Funktionen in ISRs!]
- Daraus folgt: Bei Single-Threaded Design kein Problem

Wie sieht es bei Echtzeitbetriebssystemen aus?

Portierung für RTOS

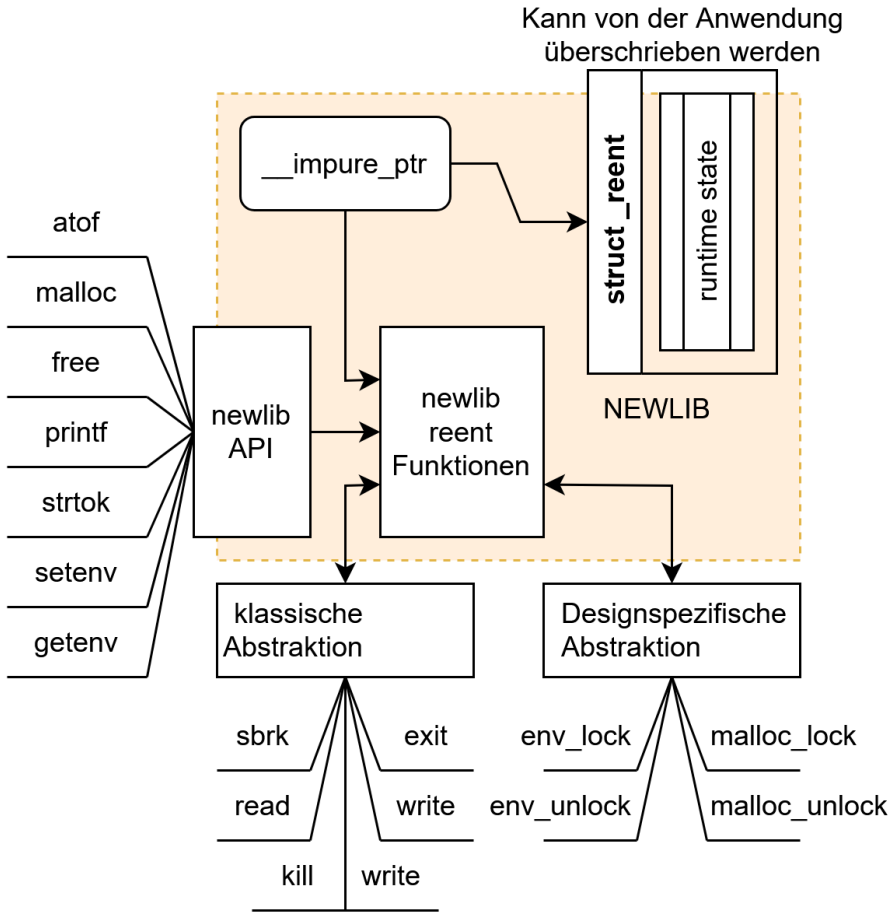
Probleme mit einem RTOS

- Durch den Scheduler ist eine Unterbrechbarkeit im System vorhanden
  - Ausschließlich Kooperatives Scheduling ist noch kein Problem
  - Präemptives- und Zeitscheiben-Scheduling sind ohne weiteres problematisch
  - ISRs sind ohnehin immer problematisch!



Technische Lösung (Abstraktion)

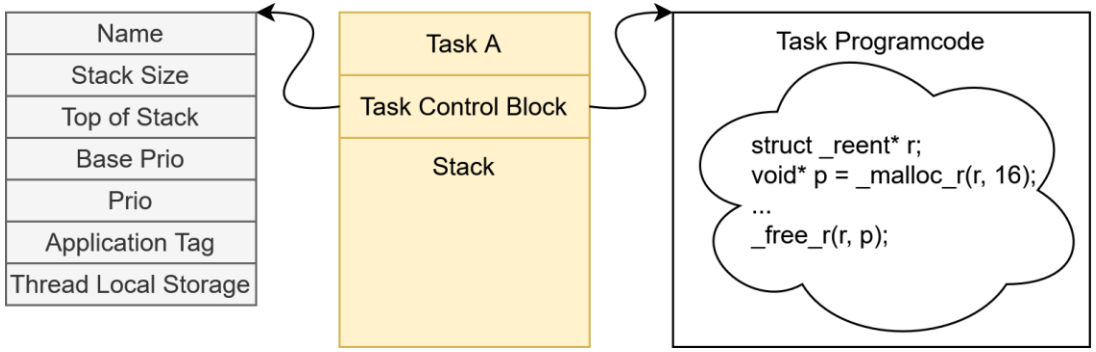
- Die newlib bietet die designspezifischen Funktionen um den gemeinsamen Speicher (malloc) und die Umgebungsvariablen (env) zu schützen
- Die klassischen Abstraktionsfunktionen können thread-safe implementiert werden, sofern notwendig



Was fehlt?

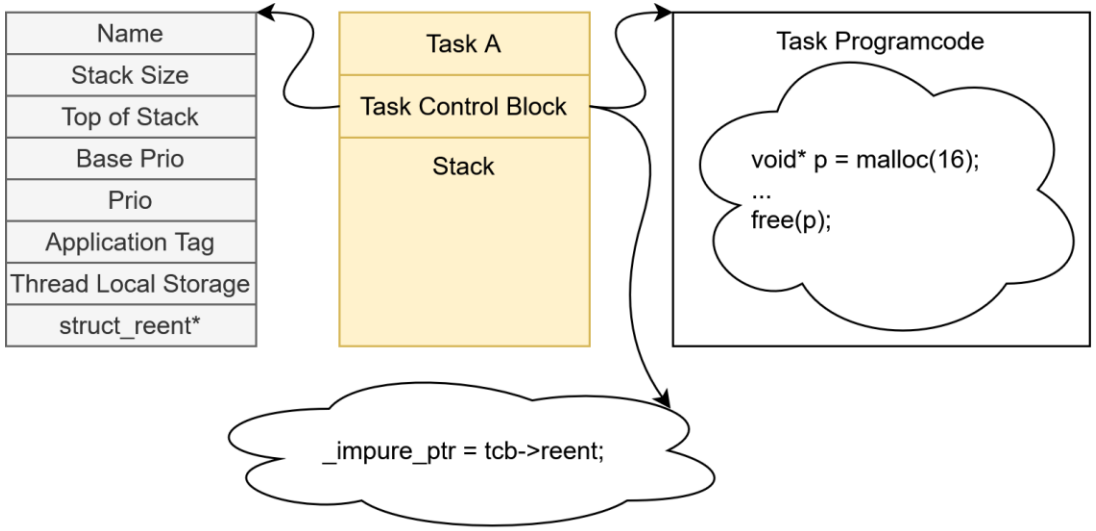
Was ist mit dem _impure_ptr?

1. Lösung



- Bietet das RTOS keine Möglichkeit, Reentrancy-Support der newlib bereitzustellen, dazu gehört:
  - Anlegen und Freigeben einer eigenen struct _reent
  - Den _impure_ptr bei jedem Kontextwechsel korrekt setzen.
- So muss die _reent-Struktur selbst bereitgestellt werden und es dürfen pro Task nur die *_r Funktionen genutzt werden.

2. Lösung



- Bietet das RTOS eine Möglichkeit, Reentrancy-Support der newlib bereitzustellen, so setzt der Scheduler beim Kontextwechsel die _reent-Struktur korrekt pro Task.
- Diese Struktur befindet sich im Task Kontrollblock des Tasks und wird auch wieder freigegeben, wenn der Task beendet wird.

Lösung 2 ist fast immer vorzuziehen!

- Weil eine echte Abstraktion der klassischen stdlib-Funktionen zur Verfügung steht.
- Dadurch ist eine höhere Portabilität möglich und der Code kann allgemeiner gehalten werden.
- Ist die 2. Lösung unsicher oder nicht umsetzbar oder reicht der RAM nicht aus, so folgt Lösung1

Noch Fragen?